# TP5 : Déterminisation accessible

**Objectifs du TP :**

- Implémenter automates déterministes et non déterministes en C.
- Implémenter l'algorithme de déterminisation accessible vu en cours.
- Réviser la notion de pointeur, de structure et l'utilisation d'un module externe.

Vous trouverez dans l'espace partagé 4 documents à copier dans votre répertoire personnel. Le module constitué de dicos.h et dicos.c ne sera utilisé que dans les parties 2 et 3. Le fichier source TP5_enonce.c contient la définition des structures utilisées, quelques fonctions de la deuxième partie et du code commenté dans le main destiné aux tests.

Le makefile permet de compiler ce source en y liant le module dicos. Taper make depuis le répertoire contenant les 4 fichiers génère un exécutable du nom de TP5 que vous pourrez exécuter comme d'habitude.

**Partie 1** *Automates finis déterministes en C*

Dans cette partie on réimplémente en C une partie des fonctions que nous avions implémenté en Ocaml la fois précédente. Un automate fini déterministe est représenté à l'aide du type suivant :

```
struct AFD {
    int taille_Q;
    int taille_Sigma;
    int q0;
    bool* finaux;
    int** delta;
};
typedef struct AFD afd;
```

Si $A = (\Sigma, Q, q_0, F, \delta)$ est un automate déterministe, on considèrera que ses états sont numérotés de 0 à $|Q|-1$ et que les lettres de $\Sigma$ sont numérotées de 0 à $|\Sigma|-1$. Ainsi, si A est de type afd et représente $A$ :

- A.taille_Q représente $|Q|$.
- A.taille_Sigma représente $|\Sigma|$.
- A.q0 représente l'état initial $q_0$.
- A.finaux est un tableau de booléens contenant true à la case q si et seulement si l'état q est final.
- A.delta représente un tableau de tableaux d'entiers tel que pour tout $q \in Q$ et tout $a \in \Sigma$, si q représente $q$ et a le numéro de la lettre $a$, A.delta[q][a] $= \begin{cases} \delta(q,a) \text{ si cet état existe} \\ -1 \text{ si on est en présence d'un blocage.} \end{cases}$

En pratique, les lettres des alphabets que nous utiliseront seront les minuscules latines. Il faudra donc au besoin savoir traduire un caractère en un entier et réciproquement. On rappelle que le caractère ASCII associé à 97 est a. Par ailleurs, nos fonctions ne manipuleront pas directement un automate mais un pointeur sur un tel objet (passage par référence plutôt que par valeur).

1. Ecrire une fonction void liberer_afd(afd* A) qui libère toute la mémoire occupée par un automate. On prendra garde à l'ordre des libérations et à bien toutes les faire.
2. Ecrire une fonction afd* initialiser_afd(int taille_Q, int taille_Sigma, int q0) qui créé (un pointeur sur) un automate contenant taille_Q états, sur un alphabet de taille_Sigma lettres, d'état initial q0 (qu'on supposera licite sans le vérifier), sans aucun état final ni aucune transition.

3. Ecrire une fonction void ajout_transition_afd(afd* A, int q, char a, int p) qui modifie l'automate (pointé par) A de sorte à ajouter la transition allant de l'état q à l'état p via la lecture de a. On rappelle que (int) a calcule l'entier associé au caractère ASCII a.

4. Décommenter dans le main les lignes correspondant à la création de l'automate $A_1$. Dessiner cet automate et identifier le langage qu'il reconnaît par la méthode de votre choix.

5. Créer un automate déterministe $A2$ reconnaissant les mots de $\{a, b\}^\star$ commençant par $a$ et dont la taille est multiple de trois.

Comme dans le TP précédent, on s'intéresse à présent à l'opération qui est la raison d'être d'un automate : la reconnaissance de mots.

6. Ecrire une fonction int delta_etoile_afd(afd* A, int q, char* u) indiquant l'état atteint en lisant le mot u dans l'automate A à partir de l'état q. La fonction renverra $-1$ en cas de blocage.

7. En déduire une fonction bool reconnu_afd(afd* A, char* u) renvoyant true si et seulement si le mot u est reconnu par l'automate déterministe A.

8. On considère les mots $u = abbabbabaab$, $v = baababbbbba$ et $w = aaababab$. Parmi ces mots, prédire lesquels sont reconnus par $A_1$, lesquels sont reconnus par $A_2$ et vérifier en décommentant les lignes idoines dans le main.

**Partie 2** *Look ma ! No determinism.*

Dans cette partie et la suivante, on s'intéresse à la manipulation d'automates non déterministes. Ces derniers sont représentés à l'aide du type afnd ci-dessous :

```
struct AFND {
    int taille_Q;
    int taille_Sigma;
    bool* initiaux;
    bool* finaux;
    liste** delta;
};
typedef struct AFND afnd;
```

Les modifications par rapport aux automates déterministes sont les suivantes :

- Les états initiaux sont à présent représentés par un tableau de booléens dont la case $q$ contient true si et seulemnt si $q$ est un des états initiaux.
- Les transitions sont représentées à l'aide d'un tableau de tableaux de listes chaînées. Plus précisément A.delta[q][a] est une liste contenant tous les états p qu'on peut atteindre en lisant a depuis q.

Le type liste est introduit dans le module dicos et est défini très classiquement comme suit :

```
struct maillon {
    int val;
    struct maillon* suivant;
};
typedef struct maillon maillon;
typedef maillon* liste;
```

Pour répondre aux questions suivantes, vous aurez besoin d'une fonction sur les listes **déjà implémentée dans le module dicos, donc que vous pouvez utiliser tel quel** : liste cons(int x, liste l) permet d'ajouter en tête l'entier x à la liste l.

Le fichier TP5_enonce.c fournit par ailleurs deux fonctions de libération et d'initialisation d'automate non déterministe similaires à celle écrites pour les questions 1 et 2. Lisez les attentivement de sorte à bien comprendre leur fonctionnement.

9. Ecrire une fonction void ajout_transition_afnd(afnd* B, int q, char a, int p) permettant de faire la même opération que la question 3 mais dans un automate non déterministe.

10. Décommenter les lignes du main concernées pour créer l'automate non déterministe $B_1$. Dessiner cet automate et identifier le langage qu'il reconnaît.

11. Créer dans le main un automate non déterministe $B_2$ simple reconnaissant les mots de $\{a, b\}^\star$ dont l'antépénultième lettre est un $a$.

La reconnaissance de mots s'adapte du cas déterministe au cas non déterministe. Dans toute la suite, y compris la partie 3, on représentera un ensemble d'états $E \subset Q$ dans un automate dont les états sont les éléments de $Q$ par un tableau de booléens dont la $q$-ème case contient true si $q \in E$ et false sinon.

12. Ecrire une fonction bool* delta_etats(afnd* B, bool* etats_depart, char a) prenant en entrée un automate non déterministe $B = (\Sigma, Q, I, F, \delta)$, un ensemble d'états $E \subset Q$ et une lettre $a \in \Sigma$ et renvoyant l'ensemble d'états $\delta(E, a) = \bigcup_{e \in E} \delta(e, a)$.

L'ensemble calculé par delta_etats est ainsi l'ensemble des états qu'on peut atteindre en lisant $a$ à partir d'un des états de $E$ dans l'automate $B$.

13. En déduire une fonction bool* delta_etoile_afnd(afnd* B, bool* etats_depart, char* u) renvoyant l'ensemble des états qu'il est possible d'atteindre dans B en lisant le mot u à partir de l'ensemble d'états etats_depart.

14. En déduire finalement une fonction bool reconnu_afnd(afnd* B, char* u) indiquant si le mot u est reconnu par l'automate fini non déterministe B.

15. Parmi les mots $u, v, w$ de la question 7, déterminer à la main ceux qui sont reconnus par $B_1$ et ceux reconnus par $B_2$. Vérifier vos conclusions avec reconnu_afnd.

**Partie 3** *Déterminé(e)s, déterminisons*

L'objectif de cette partie est d'implémenter deux algorithmes de déterminisation d'un automate à $|Q|$ états. Le premier, naïf, consiste simplement à calculer les $2^{|Q|}$ états possibles de l'automate des parties puis à y ajouter les transitions adaptées. L'automate obtenu aura donc systématiquement un nombre d'états exponentiel en celui de l'automate en entrée dont certains seront inaccessibles. Le deuxième, moins naïf, consiste en l'implémentation de l'algorithme de déterminisation accessible vu en cours.

Dans les deux cas, les états de l'automate des parties sont des ensembles d'états (représentés par des tableaux de booléens selon nos conventions) qu'il faudra donc renuméroter par des entiers dans l'automate déterministe final : l'objectif des questions qui suivent est de se doter de fonctions de conversion.

16. Ecrire une fonction int etats_vers_entier(bool* etats, int taille_Q) prenant en entrée un ensemble d'états $E \subset Q$ et l'entier $|Q|$ et renvoyant l'entier de $[\![0, 2^{|Q|} - 1]\!]$ naturellement associé à $E$.

17. Ecrire une fonction bool* entier_vers_etats(int etat, int taille_Q) réciproque de la précédente.

18. En déduire une fonction naïve afd* determiniser(afnd* B) qui construit naïvement l'automate des parties de B. Déterminiser $B_1$ via cette méthode et vérifier que l'automate obtenu reconnaît les mêmes mots de $\{u, v, w\}$ que $B_1$.

Les dernières questions visent à ne construire que les états accessibles de l'automate des parties selon l'algorithme vu en cours. Pour ce faire, on se dote de dictionnaires implémentés dans le module dicos à

l'aide de tables de hachage. Il n'est dans un premier temps pas nécessaire de s'intéresser à l'implémentation des fonctions ci-dessous pour pouvoir les utiliser :

- void liberer_dico(dico* d) permet de libérer la mémoire occupée par le dictionnaire (pointé par) d.
- dico* creer_dico(void) créé un dictionnaire vide.
- int taille_dico(dico* d) indique le nombre d'éléments stockés dans le dictionnaire d.
- bool appartient_dico_cle(dico* d, int k) renvoie true si et seulement si k est une clé d'un des éléments stockés dans le dictionnaire (pointé par) d.
- int valeur_associee(dico* d, int k) renvoie la valeur associée à la clé k dans le dictionnaire d.
- int obtenir_cle(dico* d) renvoie la clé d'un des éléments du dictionnaire d s'il n'est pas vide.
- void ajoute_entree(dico* d, int k, int v) ajoute l'association (clé valeur) (k,v) au dictionnaire d. Une erreur se produit si k est déjà la clé d'un élément présent dans le dictionnaire.
- void supprime_entree(dico* d, int k) supprime l'association (clé,valeur) (k,v) du dictionnaire d.
- liste liste_cles(dico* d) donne la liste des clés des éléments présents dans un dictionnaire d.

19. A l'aide des fonctions sur les dictionnaires, écrire une fonction dico* etats_accessibles(afnd* B) renvoyant un dictionnaire dont les clés sont les numéros associés aux ensembles d'états accessibles dans l'automate des parties (selon la numérotation de etats_vers_entier) et les valeurs sont des entiers consécutifs à partir de 0 permettant de renuméroter correctement ces états.

    Par exemple, si $\{0, 2, 3\}$ est le 5-ème état (en commençant la numérotation à 0) rencontré lors de la construction des états accessibles de l'automate des parties, on ajoutera au dictionnaire l'association $(12, 5)$ à condition qu'elle ne soit pas déjà présente. Le numéro de la partie correspondant aux états initiaux sera donc la clé de 0 dans ce dictionnaire.

20. En déduire une fonction afd* determiniser_accessible(afnd* B) qui déterminise un automate en ne considérant que les états accessibles de son automate des parties.

21. Déterminiser $B_2$ via cet algorithme et vérifier que l'automate déterministe obtenu reconnaît les mêmes mots de $\{u, v, w\}$ que $B_2$.

22. Combien d'états ont les déterminisés de $B_1$ et $B_2$ via cette méthode ? Vérifier que vous obtenez bien les mêmes résultats en déterminisant ces automates à la main.